Abstract

This document describes a method of performing multi-master communication using the I\(^2\)C bus interface (RIIC) in the RX210, RX21A, and RX220 Groups.

Products

- RX210, RX21A, and RX220 Groups

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
Contents

1. Specifications ........................................................................................................................................4
2. Operation Confirmation Conditions ............................................................................................... 5
3. Reference Application Notes ...........................................................................................................5
4. Hardware .......................................................................................................................................... 6
   4.1 Hardware Configuration ............................................................................................................. 6
   4.2 Pins Used ................................................................................................................................... 6
5. Software .......................................................................................................................................... 7
   5.1 Operation Overview .................................................................................................................. 8
   5.1.1 Master Transmission ..............................................................................................................8
   5.1.2 Master Reception ................................................................................................................9
   5.1.3 Slave Operation .................................................................................................................. 10
   5.1.3.1 Slave Transmission ....................................................................................................... 10
   5.1.3.2 Slave Reception ............................................................................................................. 10
   5.2 File Composition ..................................................................................................................... 11
   5.3 Option-Setting Memory ........................................................................................................... 11
   5.4 Constants ................................................................................................................................ 12
   5.5 Variables .................................................................................................................................. 12
   5.6 Functions .................................................................................................................................. 15
   5.7 Function Specifications ............................................................................................................. 16
   5.8 Flowcharts ................................................................................................................................ 22
   5.8.1 Main Processing .................................................................................................................. 22
   5.8.2 Port Initialization ................................................................................................................ 23
   5.8.3 Callback Function (Completion of Slave Transmission) ..................................................... 23
   5.8.4 Callback Function (Completion of Slave Reception) ........................................................ 24
   5.8.5 Callback Function (Completion of Master Transmission/Reception) ............................... 24
   5.8.6 User Interface Function (RIIC Initialization) ....................................................................... 25
   5.8.7 User Interface Function (Start a Slave Operation) ........................................................... 26
   5.8.8 User Interface Function (Start a Master Operation) .......................................................... 27
   5.8.9 User Interface Function (Obtain the Master State) ............................................................ 28
   5.8.10 Enabling the RIIC .............................................................................................................. 29
   5.8.11 Disabling the RIIC ............................................................................................................ 30
   5.8.12 Enabling the RIIC Interrupts ............................................................................................ 31
   5.8.13 Disabling the RIIC Interrupts ............................................................................................ 32
   5.8.14 Receive Data Full Interrupt ............................................................................................... 33
   5.8.15 Transmit Data Empty Interrupt ...................................................................................... 35
   5.8.16 Transmit End Interrupt .................................................................................................... 36
   5.8.17 Stop Condition Detection Interrupt ................................................................................. 37
   5.8.18 NACK Detection Interrupt ............................................................................................... 38
   5.8.19 Arbitration-Lost Detection Interrupt .............................................................................. 39
   5.8.20 Timeout Detection Interrupt ............................................................................................ 39
   5.8.21 Start Condition Detection Interrupt ............................................................................... 39
   5.8.22 RIIC0.EEI0 Interrupt Handling ...................................................................................... 40
6. Applying This Application Note to the RX21A or RX220 Group ....................................................... 42

7. Sample Code .................................................................................................................................... 43

8. Reference Documents ...................................................................................................................... 43
1. Specifications

Multi-master communication with \(\text{I}^2\text{C}\) bus is performed using the RIIC.

After a reset, a master transmission and master reception are performed once each. 10-byte data (00h to 09h) is transmitted in a master transmission, then 10-byte data is received in a master reception.

When an arbitration-lost is detected during a master transmission or master reception, a slave operation is performed while other master device communications are prioritized.

- Transfer rate: 100 kbps
- Address format: 7-bit address format
- Master/slave operations: Master transmission, master reception, slave transmission, and slave reception

Refer to the User’s Manual: Hardware for the product used and the \(\text{I}^2\text{C}\) bus specifications for details on \(\text{I}^2\text{C}\) bus communication formats.

Table 1.1 lists the Peripheral Function and Its Application and Figure 1.1 shows the Operation Overview.

### Table 1.1 Peripheral Function and Its Application

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIIC</td>
<td>Multi-master (\text{I}^2\text{C}) bus</td>
</tr>
</tbody>
</table>

![Figure 1.1 Operation Overview](image-url)
2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>R5F52108ADFP (RX210 Group)</td>
</tr>
<tr>
<td>Operating frequencies</td>
<td>- Main clock: 20 MHz</td>
</tr>
<tr>
<td></td>
<td>- PLL: 100 MHz (main clock divided by 2 and multiplied by 10)</td>
</tr>
<tr>
<td></td>
<td>- System clock (ICLK): 50 MHz (PLL divided by 2)</td>
</tr>
<tr>
<td></td>
<td>- Peripheral module clock B (PCLKB): 25 MHz (PLL divided by 4)</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5.0 V</td>
</tr>
<tr>
<td>Integrated development</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td>environment</td>
<td>High-performance Embedded Workshop Version 4.09.01</td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics Corporation</td>
</tr>
<tr>
<td></td>
<td>C/C++ Compiler Package for RX Family V.1.02 Release 01</td>
</tr>
<tr>
<td>Compile options</td>
<td>-cpu=rx200 -output=obj=&quot;$(CONFIGDIR)&quot;$(FILELEAF).obj&quot; -debug -nologo</td>
</tr>
<tr>
<td></td>
<td>(The default setting is used in the integrated development environment.)</td>
</tr>
<tr>
<td>iodfie.h version</td>
<td>Version 1.2A</td>
</tr>
<tr>
<td>Endian</td>
<td>Little endian</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Single-chip mode</td>
</tr>
<tr>
<td>Processor mode</td>
<td>Supervisor mode</td>
</tr>
<tr>
<td>Sample code version</td>
<td>Version 1.00</td>
</tr>
<tr>
<td>Board used</td>
<td>Renesas Starter Kit for RX210 (product part no.: R0K505210C000BE)</td>
</tr>
</tbody>
</table>

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RX210 Group Initial Setting Rev. 2.00 (R01AN1002EJ)
- RX21A Group Initial Setting Rev. 1.10 (R01AN1486EJ)
- RX220 Group Initial Setting Rev. 1.10 (R01AN1494EJ)

The initial setting functions in the reference application notes are used in the sample code in this application note. The revision numbers of the reference application notes are current as of when this application note was made. However the latest version is always recommended. Visit the Renesas Electronics Corporation website to check and download the latest version.
4. Hardware

4.1 Hardware Configuration

Figure 4.1 shows a Connection Example.

![Figure 4.1 Connection Example](image)

4.2 Pins Used

Table 4.1 lists the Pins Used and Their Functions.

The pins described here are for 100-pin products. When the product with less than 100-pin is used, select appropriate pins for the product used.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P12/SCL</td>
<td>I/O</td>
<td>Serial clock I/O pin</td>
</tr>
<tr>
<td>P13/SDA</td>
<td>I/O</td>
<td>Serial data I/O pin</td>
</tr>
</tbody>
</table>
5. Software

After a reset, perform the initialization to enable the RIIC operation and start the slave operation.

After confirming the IIC bus is in the bus free state, start 10-byte master transmission with 20h as the slave address, 0 as the R/W bit value, and 00h to 09h as the transmit data. When the master transmission is completed, confirm the IIC bus is in the bus free state, and start 10-byte master reception with 20h as the slave address and 1 as the R/W bit value. When the master reception is completed, confirm the IIC bus is in the bus free state, and disable the RIIC operation.

When a master arbitration-lost is detected, the operation is automatically switched to slave reception mode. If the slave addresses match, a slave reception or slave transmission is performed according to the R/W bit setting.

When a NACK is detected, a transfer operation is canceled automatically. Then generate a stop condition and terminate the communication.

After a master transmission is started, when the following conditions are met, the callback function is called.

- An arbitration-lost is detected.
- Master transmission is completed.
- Master reception is completed.

Settings for peripheral functions are as follows:

RIIC

- Master/slave operations: Master transmission, master reception, slave reception, and slave transmission
- Address format: 7-bit address format
- Slave address: 10h
- Transfer rate: 100 kbps
- Arbitration-lost detection: Master arbitration-lost detection
- Interrupts: Transmit data empty interrupt (ICTXI) enabled
  Transmit end interrupt (ICTEI) enabled
  Receive data full interrupt (ICRXI) enabled
  NACK reception interrupt (NAKI) enabled
  Stop condition detection interrupt (SPI) enabled
  Arbitration-lost interrupt (ALI) enabled
5.1 Operation Overview

5.1.1 Master Transmission

(1) Start master transmission
Verify the ICCR2.BBSY flag is 0, then set the ICCR2.ST bit to 1 (requests to issue a start condition).

(2) Issue a start condition
When a start condition is issued, the ICSR2.TDRE flag becomes 1, and a TXI0 interrupt request is generated. In the TXI0 interrupt handling, write the slave address and value of the R/W# bit to the ICDRT register.

(3) Transmit data
When data is transferred from the ICDRT register to the ICDRS register, the TDRE flag becomes 1 again, and a TXI0 interrupt request is generated. Write the value of the master transmit buffer to the ICDRT register in the TXI0 interrupt handling. When the last data is written, in the subsequent TXI0 interrupt handling, set the ICIER.TIE bit to 0 (transmit data empty interrupt request (ICTXI) is disabled) and the ICIER.TEIE bit to 1 (transmit end interrupt request (ICTEI) is enabled).

(4) Complete the transmission
When the last data transmission is completed, the ICSR2.TEND flag becomes 1, and a TEI0 interrupt request is generated. Set the ICCR2.SP bit to 1 (requests to issue a stop condition) in the TEI0 interrupt handling.

(5) Issue a stop condition
When a stop condition is issued, the ICSR2.STOP flag becomes 1, and the EEI0 interrupt request is generated. In the EEI0 interrupt handling, set the TIE bit to 1, set the TEIE bit to 0, and call the callback function (completion of master transmission/reception).

Figure 5.1 shows the Timing Diagram of Master Transmission.
5.1.2 Master Reception

(1) Start master reception
Verify the ICCR2.BBSY flag is 0, then set the ICCR2.ST bit to 1 (requests to issue a start condition).

(2) Issue a start condition
When a start condition is issued, the ICSR2.TDRE flag becomes 1, and a TXI0 interrupt request is generated. In the TXI0 interrupt handling, write the slave address and value of the R/W# bit to the ICDRT register. Set the ICIER.TIE bit to 0 (transmit data empty interrupt request (ICTXI) is disabled) and the IR flag of the TXI0 interrupt to 0.

(3) Complete the slave address transmission
When the transmission for the slave address is completed, the ICSR2.RDRF flag becomes 1, and a RXI0 interrupt request is generated. Dummy read the ICDRR register in the RXI0 interrupt handling.

(4) Complete the reception
When the data reception is completed, the RDRF flag becomes 1, and a RXI0 interrupt request is generated. In the RXI0 interrupt handling, the ICDRR register value is stored in the master receive buffer. When the reception for third to last byte of data is completed, set the ICMR3.WAIT bit to 1. When the reception for second to last byte of data is completed, set the ICMR3.RDRFS bit to 1 and the ICMR3.ACKBT bit to 1. When the last data reception is completed, set the ICCR2.SP bit to 1 (requests to issue a stop condition), the ACKBT bit to 1, and the WAIT bit to 0.

(5) Issue a stop condition
When a stop condition is issued, the ICSR2.STOP flag becomes 1, and the EEI0 interrupt request is generated. In the EEI0 interrupt handling, set the TIE bit to 1 and call the callback function (completion of master transmission/reception).

Figure 5.2 shows the Timing Diagram of Master Reception.
5.1.3 Slave Operation
When slave addresses match during a slave operation, a slave transmission or slave reception is started according to the R/W bit. When a slave transmission or slave reception is completed, stop the slave operation and call the callback function. When continuing the slave operation, call the user interface function (start a slave operation) again. The sample code calls the user interface function (start a slave operation) with the callback function to continue the slave operation.

5.1.3.1 Slave Transmission
- After a slave transmission is started, transmit data in the slave transmit buffer from the top until a NACK is received.
- Transmit FFh when the number of transmissions exceeds the specified byte count of slave transmissions.
- When the transmission is completed, call the callback function. However if a NACK is received while the number of transmissions is less than the specified byte count of slave transmissions, the callback function is not called. When the slave transmission is started again, transmit data in the slave transmit buffer from the top.

5.1.3.2 Slave Reception
- After a slave reception is started, store received data in the slave receive buffer from the top until a stop condition is detected.
- Received data which exceeds the specified byte count of slave receptions is discarded without being stored in the slave receive buffer.
- When a reception is completed, call the callback function. However if a stop condition is detected while the number of receptions is less than the specified byte count of slave receptions, the callback function is not called. When the slave reception is started again, store data in the slave receive buffer from the top (overwrite data if it already exists).
5.2 File Composition

Table 5.1 lists the Files Used in the Sample Code. Files generated by the integrated development environment are not included in this table.

Table 5.1 Files Used in the Sample Code

<table>
<thead>
<tr>
<th>File Name</th>
<th>Outline</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Main processing</td>
<td></td>
</tr>
<tr>
<td>r_init_stop_module.c</td>
<td>Stop processing for active peripheral functions after a reset</td>
<td></td>
</tr>
<tr>
<td>r_init_stop_module.h</td>
<td>Header file for r_init_stop_module.c</td>
<td></td>
</tr>
<tr>
<td>r_init_non_existent_port.c</td>
<td>Nonexistent port initialization</td>
<td></td>
</tr>
<tr>
<td>r_init_non_existent_port.h</td>
<td>Header file for r_init_non_existent_port.c</td>
<td></td>
</tr>
<tr>
<td>r_init_clock.c</td>
<td>Clock initialization</td>
<td></td>
</tr>
<tr>
<td>r_init_clock.h</td>
<td>Header file for r_init_clock.c</td>
<td></td>
</tr>
<tr>
<td>rlic.c</td>
<td>RIIC processing</td>
<td></td>
</tr>
<tr>
<td>rlic_int.c</td>
<td>RIIC interrupt handling</td>
<td></td>
</tr>
<tr>
<td>rlic.h</td>
<td>Header file for rlic.c</td>
<td></td>
</tr>
</tbody>
</table>

5.3 Option-Setting Memory

Table 5.2 lists the Option-Setting Memory Used in the Sample Code. When necessary, set a value suited to the user system.

Table 5.2 Option-Setting Memory Used in the Sample Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFS0</td>
<td>FFFF FF8Fh to FFFF FF8Ch</td>
<td>FFFF FFFFh</td>
<td>The IWDT is stopped after a reset. The WDT is stopped after a reset.</td>
</tr>
<tr>
<td>OFS1</td>
<td>FFFF FF8Bh to FFFF FF88h</td>
<td>FFFF FFFFh</td>
<td>The voltage monitor 0 reset is disabled after a reset. HOCO oscillation is disabled after a reset.</td>
</tr>
<tr>
<td>MDES</td>
<td>FFFF FF83h to FFFF FF80h</td>
<td>FFFF FFFFh</td>
<td>Little endian</td>
</tr>
</tbody>
</table>
5.4 Constants

Table 5.3 lists the Constants Used in the Sample Code.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELF_ADDRESS</td>
<td>10h</td>
<td>Self-slave address</td>
</tr>
<tr>
<td>SLV_ADDRESS</td>
<td>20h</td>
<td>Slave address</td>
</tr>
<tr>
<td>MST_DATA_NUM</td>
<td>(10 + 1)</td>
<td>Byte count of master transmissions/receptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(byte count + slave address)</td>
</tr>
<tr>
<td>SLV_DATA_NUM</td>
<td>10</td>
<td>Byte count of slave transmissions/receptions</td>
</tr>
<tr>
<td>MST_WRITE</td>
<td>00h</td>
<td>Set value of the R/W bit: 0: Write</td>
</tr>
<tr>
<td>MST_READ</td>
<td>01h</td>
<td>Set value of the R/W bit: 1: Read</td>
</tr>
<tr>
<td>RIIC_OK</td>
<td>1</td>
<td>Return value of the RiicMstStart function</td>
</tr>
<tr>
<td>RIIC_NG</td>
<td>2</td>
<td>Return value of the RiicMstStart function</td>
</tr>
<tr>
<td>RIIC_BUS_BUSY</td>
<td>0</td>
<td>Return value of the RiicGetMstState function</td>
</tr>
<tr>
<td>RIIC_ST_MST_STOP</td>
<td>1</td>
<td>Return value of the RiicGetMstState function</td>
</tr>
<tr>
<td>RIIC_ST_MST_BUSY</td>
<td>2</td>
<td>Return value of the RiicGetMstState function</td>
</tr>
<tr>
<td>RIIC_ST_MST_NACK</td>
<td>3</td>
<td>Return value of the RiicGetMstState function</td>
</tr>
<tr>
<td>RIIC_ST_MST_AL</td>
<td>4</td>
<td>Return value of the RiicGetMstState function</td>
</tr>
<tr>
<td>RIIC_SET</td>
<td>1</td>
<td>Set the flag.</td>
</tr>
<tr>
<td>RIIC_CLEAR</td>
<td>0</td>
<td>Clear the flag.</td>
</tr>
<tr>
<td>RIIC_ENABLE</td>
<td>1</td>
<td>Enable the RIIC.</td>
</tr>
<tr>
<td>RIIC_DISABLE</td>
<td>0</td>
<td>Disable the RIIC.</td>
</tr>
<tr>
<td>RIIC_RXI</td>
<td>01h</td>
<td>Argument of the RiicIntEna and RiicIntDis functions</td>
</tr>
<tr>
<td>RIIC_TXI</td>
<td>02h</td>
<td>Argument of the RiicIntEna and RiicIntDis functions</td>
</tr>
<tr>
<td>RIIC_TEI</td>
<td>04h</td>
<td>Argument of the RiicIntEna and RiicIntDis functions</td>
</tr>
</tbody>
</table>

5.5 Variables

Table 5.4 lists the Global Variables, and Table 5.5 and Table 5.6 list the static Variables.

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>MstTrmBuff[256]</td>
<td>Master transmit buffer</td>
<td>main</td>
</tr>
<tr>
<td>uint8_t</td>
<td>MstRcvBuff[256]</td>
<td>Master receive buffer</td>
<td>main</td>
</tr>
<tr>
<td>uint8_t</td>
<td>SlvTrmBuff[256]</td>
<td>Slave transmit buffer</td>
<td>main</td>
</tr>
<tr>
<td>uint8_t</td>
<td>SlvRcvBuff[256]</td>
<td>Slave receive buffer</td>
<td>main</td>
</tr>
</tbody>
</table>
### Table 5.5 static Variables (1/2)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>static uint8_t</td>
<td>RiicTrmAddr</td>
<td>Slave address</td>
<td>RiicMstStart RiicTDRE</td>
</tr>
<tr>
<td>static uint8_t*</td>
<td>RiicMstBuff</td>
<td>Pointer to a master transmit/receive buffer</td>
<td>RiicMstStart RiicRDRF RiicTDRE RiicTEND</td>
</tr>
<tr>
<td>static uint32_t</td>
<td>RiicMstCnt</td>
<td>Master transmission/reception counter</td>
<td>RiicIni</td>
</tr>
<tr>
<td>static uint32_t</td>
<td>RiicMstNum</td>
<td>Byte count of master transmissions/receptions</td>
<td>RiicMstStart RiicRDRF RiicTDRE RiicTEND</td>
</tr>
<tr>
<td>static uint8_t*</td>
<td>RiicSlvTrmBuff</td>
<td>Pointer to a slave transmit buffer</td>
<td>RiicSlvStart RiicTDRE RiicSTOP</td>
</tr>
<tr>
<td>static uint8_t*</td>
<td>RiicSlvTrmStartBuff</td>
<td>Pointer to a slave transmit buffer</td>
<td>RiicSlvStart RiicSTOP</td>
</tr>
<tr>
<td>static uint32_t</td>
<td>RiicSlvTrmCnt</td>
<td>Slave transmission counter</td>
<td>RiicIni RiicSlvStart RiicTDRE RiicTEND RiicSTOP</td>
</tr>
<tr>
<td>static uint8_t*</td>
<td>RiicSlvRcvBuff</td>
<td>Pointer to a slave receive buffer</td>
<td>RiicSlvStart RiicRDRF RiicSTOP</td>
</tr>
<tr>
<td>static uint8_t*</td>
<td>RiicSlvRcvStartBuff</td>
<td>Pointer to a slave receive buffer</td>
<td>RiicSlvStart RiicSTOP</td>
</tr>
<tr>
<td>static uint32_t</td>
<td>RiicSlvRcvCnt</td>
<td>Slave reception counter</td>
<td>RiicIni RiicSlvStart RiicRDRF RiicSTOP</td>
</tr>
<tr>
<td>static uint32_t</td>
<td>RiicSlvNum</td>
<td>Byte count of slave transmissions/receptions</td>
<td>RiicIni RiicSlvStart RiicRDRF RiicTDRE RiicTEND RiicSTOP</td>
</tr>
</tbody>
</table>
### Table 5.6 static Variables (2/2)

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>static volatile uint8_t</td>
<td>RiicStartFlg</td>
<td>Communication flag:</td>
<td>RiicIni, RiicRDRF, RiicTDRE, RiicSTOP, RiicAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Not communicating</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Communicating</td>
<td></td>
</tr>
<tr>
<td>static volatile uint8_t</td>
<td>RiicMstFlg</td>
<td>Master operation flag:</td>
<td>RiicMstStart, RiicRDRF, RiicTDRE, RiicTEND, RiicSTOP, RiicNACK, RiicAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Slave operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Master operation</td>
<td></td>
</tr>
<tr>
<td>static volatile uint8_t</td>
<td>RiicMstState</td>
<td>Master state:</td>
<td>RiicIni, RiicMstStart, RiicGetMstState, RiicSTOP, RiicNACK, RiicAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Stopped</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Busy (communicating)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: NACK detected</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: Arbitration-lost detected</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: Communication completed</td>
<td></td>
</tr>
</tbody>
</table>
### 5.6 Functions

Table 5.7 lists the Functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>port_init</td>
<td>Port initialization</td>
</tr>
<tr>
<td>R_INIT_StopModule</td>
<td>Stop processing for active peripheral functions after a reset</td>
</tr>
<tr>
<td>R_INIT_NonExistentPort</td>
<td>Nonexistent port initialization</td>
</tr>
<tr>
<td>R_INIT_Clock</td>
<td>Clock initialization</td>
</tr>
<tr>
<td>CbSlaveTrm</td>
<td>Callback function (completion of slave transmission)</td>
</tr>
<tr>
<td>CbSlaveRcv</td>
<td>Callback function (completion of slave reception)</td>
</tr>
<tr>
<td>CbMaster</td>
<td>Callback function (completion of master transmission/reception)</td>
</tr>
<tr>
<td>RiicIni</td>
<td>User interface function (RIIC initialization)</td>
</tr>
<tr>
<td>RiicSlvStart</td>
<td>User interface function (start a slave operation)</td>
</tr>
<tr>
<td>RiicMstStart</td>
<td>User interface function (start a master operation)</td>
</tr>
<tr>
<td>RiicGetMstState</td>
<td>User interface function (obtain the master state)</td>
</tr>
<tr>
<td>RiicEnable</td>
<td>Enabling the RIIC</td>
</tr>
<tr>
<td>RiicDisable</td>
<td>Disabling the RIIC</td>
</tr>
<tr>
<td>RiicIntEna</td>
<td>Enabling the RIIC interrupts</td>
</tr>
<tr>
<td>RiicIntDis</td>
<td>Disabling the RIIC interrupts</td>
</tr>
<tr>
<td>RiicRDRF</td>
<td>Receive data full interrupt</td>
</tr>
<tr>
<td>RiicTDRE</td>
<td>Transmit data empty interrupt</td>
</tr>
<tr>
<td>RiicTEND</td>
<td>Transmit end interrupt</td>
</tr>
<tr>
<td>RiicSTOP</td>
<td>Stop condition detection interrupt</td>
</tr>
<tr>
<td>RiicNACK</td>
<td>NACK detection interrupt</td>
</tr>
<tr>
<td>RiicAL</td>
<td>Arbitration-lost detection interrupt</td>
</tr>
<tr>
<td>RiicTMO</td>
<td>Timeout detection interrupt</td>
</tr>
<tr>
<td>RiicSTART</td>
<td>Start condition detection interrupt</td>
</tr>
<tr>
<td>Excep_RIIC0_EEI0</td>
<td>RIIC0.EEI0 interrupt handling</td>
</tr>
<tr>
<td>Excep_RIIC0_RXI0</td>
<td>RIIC0.RXI0 interrupt handling</td>
</tr>
<tr>
<td>Excep_RIIC0_TXI0</td>
<td>RIIC0.TXI0 interrupt handling</td>
</tr>
<tr>
<td>Excep_RIIC0_TEI0</td>
<td>RIIC0.TEI0 interrupt handling</td>
</tr>
</tbody>
</table>
### 5.7 Function Specifications

The following tables list the sample code function specifications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Outline</th>
<th>Header</th>
<th>Declaration</th>
<th>Description</th>
<th>Arguments</th>
<th>Return Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>main</strong></td>
<td>Main processing</td>
<td></td>
<td></td>
<td></td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><strong>port_init</strong></td>
<td>Port initialization</td>
<td></td>
<td></td>
<td></td>
<td>None</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><strong>R_INIT_StopModule</strong></td>
<td>Stop processing for active peripheral functions after a reset</td>
<td>r_init_stop_module.h</td>
<td>void R_INIT_StopModule(void)</td>
<td>Configure the setting to enter the module-stop state.</td>
<td>None</td>
<td>None</td>
<td>Transition to the module-stop state is not performed in the sample code. For details on this function, refer to the Initial Setting application note for the product used.</td>
</tr>
<tr>
<td><strong>R_INIT_NonExistentPort</strong></td>
<td>Nonexistent port initialization</td>
<td>r_init_non_existent_port.h</td>
<td>void R_INIT_NonExistentPort(void)</td>
<td>Initialize port direction registers for ports that do not exist in products with less than 100 pins.</td>
<td>None</td>
<td>None</td>
<td>The number of pins in the sample code is set for the 100-pin package (PIN_SIZE=100). After this function is called, when writing in byte units to the PDR registers or PODR registers which have nonexistent ports, set the corresponding bits for nonexistent ports as follows: set the I/O select bits in the PDR registers to 1 and set the output data store bits in the PODR registers to 0. For details on this function, refer to the Initial Setting application note for the product used.</td>
</tr>
</tbody>
</table>
## R_INIT_Clock

<table>
<thead>
<tr>
<th>Outline</th>
<th>Clock initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_init_clock.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_INIT_Clock(void)</td>
</tr>
<tr>
<td>Description</td>
<td>Initialize the clock.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
<tr>
<td>Remarks</td>
<td>The sample code selects processing which uses PLL as the system clock without using the sub-clock. For details on this function, refer to the Initial Setting application note for the product used.</td>
</tr>
</tbody>
</table>

## CbSlaveTrm

<table>
<thead>
<tr>
<th>Outline</th>
<th>Callback function (completion of slave transmission)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void CbSlaveTrm(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called when a slave transmission is completed.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

## CbSlaveRcv

<table>
<thead>
<tr>
<th>Outline</th>
<th>Callback function (completion of slave reception)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void CbSlaveRcv(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called when a slave reception is completed.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

## CbMaster

<table>
<thead>
<tr>
<th>Outline</th>
<th>Callback function (completion of master transmission/reception)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void CbMaster(void)</td>
</tr>
</tbody>
</table>
| Description      | This function is called when the following conditions are met after a master communication is started.  
|                  | - An arbitration-lost is detected.                             |
|                  | - A master transmission is completed.                          |
|                  | - A master reception is completed.                             |
| Arguments        | None                                                           |
| Return Value     | None                                                           |
| Remarks          | The processing of this function is not included in the sample code. Add a program as required. |
### RiicIni

**Outline**
User interface function (RIIC initialization)

**Header**
riic.h

**Declaration**
void RiicIni(uint8_t in_SelfAddr, uint8_t in_Enable)

**Description**
Initialize the RIIC.

**Arguments**
- `uint8_t in_SelfAddr`: Self-address (bit 0 is set to 0)
- `uint8_t in_Enable`: RIIC enabled/disabled:
  - RIIC_ENABLE: RIIC enabled
  - RIIC_DISABLE: RIIC disabled

**Return Value**
None

### RiicSlvStart

**Outline**
User interface function (start a slave operation)

**Header**
riic.h

**Declaration**
void RiicSlvStart(uint8_t * in_RcvAddr, uint8_t * in_TrmAddr, uint32_t in_num,
CallBackFunc cbTrm, CallBackFunc cbRcv)

**Description**
Start a slave operation.

**Arguments**
- `uint8_t * in_RcvAddr`: Pointer to a slave receive data storage
- `uint8_t * in_TrmAddr`: Pointer to a slave transmit data storage
- `uint32_t in_num`: Byte count of slave transmissions/receptions
- `CallBackFunc cbTrm`: Callback function (completion of a slave transmission)
- `CallBackFunc cbRcv`: Callback function (completion of a slave reception)

**Return Value**
None

### RiicMstStart

**Outline**
User interface function (start a master operation)

**Header**
riic.h

**Declaration**
uint8_t RiicMstStart(uint8_t in_addr, uint8_t * in_buff, uint32_t in_num,
CallBackFunc cb)

**Description**
Start a master operation.

**Arguments**
- `uint8_t in_addr`: Slave address (bit 0 is the R/W bit)
- `uint8_t * in_buff`: Pointer to a master transmit/receive data storage
- `uint32_t in_num`: Byte count of master transmissions/receptions
- `CallBackFunc cb`: Callback function (completion of a master transmission/reception)

**Return Value**
- RIIC_OK: Completed successfully
- RIIC_NG: Argument error (byte count of transmissions/receptions is less than 2)
- RIIC_BUS_BUSY: Bus busy

### RiicGetMstState

**Outline**
User interface function (obtain the master state)

**Header**
riic.h

**Declaration**
uint8_t RiicGetMstState(void)

**Description**
Return the master state.

**Arguments**
None

**Return Value**
- RIIC_ST_MST_STOP: Stopped
- RIIC_ST_MST_BUSY: Busy (communicating)
- RIIC_ST_MST_NACK: NACK detected
- RIIC_ST_MST_AL: Arbitration-lost detected
- RIIC_ST_MST_COMPLETE: Communication completed
### RiicEnable

**Outline**
Enabling the RIIC

**Header**
None

**Declaration**
void RiicEnable(uint8_t addr)

**Description**
Enable the RIIC operation.

**Arguments**
uint8_t addr  
Self-address

**Return Value**
None

---

### RiicDisable

**Outline**
Disabling the RIIC

**Header**
None

**Declaration**
void RiicDisable(void)

**Description**
Disable the RIIC operation.

**Arguments**
None

**Return Value**
None

---

### RiicIntEna

**Outline**
Enabling the RIIC interrupts

**Header**
None

**Declaration**
void RiicIntEna(uint8_t req)

**Description**
Enable the RIIC interrupt request.

**Arguments**
uint8_t req  
Requests enabled:
- RIIC_RXI: Enable the RXI interrupt request
- RIIC_TXI: Enable the TXI interrupt request
- RIIC_TEI: Enable the TEI interrupt request

**Return Value**
None

---

### RiicIntDis

**Outline**
Disabling the RIIC interrupts

**Header**
None

**Declaration**
void RiicIntDis(uint8_t req)

**Description**
Disable the RIIC interrupt request.

**Arguments**
uint8_t req  
Requests disabled:
- RIIC_RXI: Disable the RXI interrupt request
- RIIC_TXI: Disable the TXI interrupt request
- RIIC_TEI: Disable the TEI interrupt request

**Return Value**
None

---

### RiicRDRF

**Outline**
Receive data full interrupt

**Header**
riic.h

**Declaration**
void RiicRDRF(void)

**Description**
This function is called from the RIIC0.RXI0 interrupt handling and reads the receive data.

**Arguments**
None

**Return Value**
None
### RiicTDRE

<table>
<thead>
<tr>
<th>Outline</th>
<th>Transmit data empty interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>riic.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void RiicTDRE(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called from the RIIC0.TXI0 interrupt handling and writes the transmit data.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### RiicTEND

<table>
<thead>
<tr>
<th>Outline</th>
<th>Transmit end interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>riic.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void RiicTEND(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called from the RIIC0.TE10 interrupt handling and issues a stop condition.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### RiicSTOP

<table>
<thead>
<tr>
<th>Outline</th>
<th>Stop condition detection interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>riic.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void RiicSTOP(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called from the RIIC0.EE10 interrupt handling and calls the callback function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### RiicNACK

<table>
<thead>
<tr>
<th>Outline</th>
<th>NACK detection interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>riic.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void RiicNACK(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called from the RIIC0.EE10 interrupt handling and issues a stop condition.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

### RiicAL

<table>
<thead>
<tr>
<th>Outline</th>
<th>Arbitration-lost detection interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>riic.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void RiicAL(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called from the RIIC0.EE10 interrupt handling and calls the callback function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>
RiicTMO
Outline  Timeout detection interrupt
Header riic.h
Declaration void RiicTMO(void)
Description This function is called from the RIIC0.EEI0 interrupt handling.
Arguments None
Return Value None
Remarks The processing of this function is not included in the sample code. Add a program as required.

RiicSTART
Outline  Start condition detection interrupt
Header None
Declaration void RiicSTART(void)
Description This function is called from the RIIC0.EEI0 interrupt handling.
Arguments None
Return Value None
Remarks The processing of this function is not included in the sample code. Add a program as required.

Excep_RIIC0_EEI0
Outline  RIIC0.EEI0 interrupt handling
Header None
Declaration void Excep_RIIC0_EEI0(void)
Description This function performs interrupt handling when a communication error or event occurs.
Arguments None
Return Value None

Excep_RIIC0_RXI0
Outline  RIIC0.RXI0 interrupt handling
Header None
Declaration void Excep_RIIC0_RXI0(void)
Description This function performs receive data full interrupt handling.
Arguments None
Return Value None

Excep_RIIC0_TXI0
Outline  RIIC0.TXI0 interrupt handling
Header None
Declaration void Excep_RIIC0_TXI0(void)
Description This function performs transmit data empty interrupt handling.
Arguments None
Return Value None

Excep_RIIC0_TEI0
Outline  RIIC0.TEI0 interrupt handling
Header None
Declaration void Excep_RIIC0_TEI0(void)
Description This function performs transmit end interrupt handling.
Arguments None
Return Value None
5.8 Flowcharts

5.8.1 Main Processing

Figure 5.3 shows the Main Processing.

![Flowchart for Main Processing]

Figure 5.3 Main Processing
5.8.2 Port Initialization

Figure 5.4 shows the Port Initialization.

```
PORT1.PDR register
B2 bit ← 0: SCL: Input
B3 bit ← 0: SDA: Input

PORT1.PMR register
B2 bit ← 0: SCL: Use pin as general I/O port
B3 bit ← 0: SDA: Use pin as general I/O port

MPC.PWPR register
B0WI bit ← 0

MPC.PWPR register
PFSWE bit ← 1

MPC.P12PFS register
PSEL[3:0] bits ← 1111b: SCL

MPC.P13PFS register
PSEL[3:0] bits ← 1111b: SDA

MPC.PWPR register
PFSWE bit ← 0

MPC.PWPR register
B0WI bit ← 1
```

5.8.3 Callback Function (Completion of Slave Transmission)

Figure 5.5 shows the Callback Function (Completion of Slave Transmission).

```
User interface function
(start slave operation)
RiicSlvStart()
```

Figure 5.5 Callback Function (Completion of Slave Transmission)
### 5.8.4 Callback Function (Completion of Slave Reception)

Figure 5.6 shows the Callback Function (Completion of Slave Reception).

![Callback Function (Completion of Slave Reception)](image)

#### 5.8.5 Callback Function (Completion of Master Transmission/Reception)

Figure 5.7 shows the Callback Function (Completion of Master Transmission/Reception).

![Callback Function (Completion of Master Transmission/Reception)](image)

**Note:**
1. The processing of this function is not included in the sample code. Add a program as required.
5.8.6 User Interface Function (RIIC Initialization)

Figure 5.8 shows the User Interface Function (RIIC Initialization).

![Diagram of User Interface Function (RIIC Initialization)]

Arguments:
- uint8_t in_SelfAddr: Self-address
- uint8_t in_Enable: RIIC enabled or disabled

RIIC enabled?

Yes

RIIC enabled

RiicEnable()

No

RIIC disabled

RiicDisable()

return
5.8.7 User Interface Function (Start a Slave Operation)

Figure 5.9 shows the User Interface Function (Start a Slave Operation).

Arguments:
- `uint8_t * in_RcvAddr`: Pointer to a slave receive data storage
- `uint8_t * in_TrmAddr`: Pointer to a slave transmit data storage
- `uint32_t in_num`: Byte count of slave transmissions/receptions
- `CallBackFunc cbTrm`: Callback function (completion of a slave transmission)
- `CallBackFunc cbRcv`: Callback function (completion of a slave reception)

```
RiicSlvStart

Set arguments in the RAM

RiicSlvTrmStartBuff ← in_TrmAddr
RiicSlvRcvStartBuff ← in_RcvAddr
RiicSlvTrmBuff ← in_TrmAddr
RiicSlvRcvBuff ← in_RcvAddr
RiicSlvTrmCnt ← 0
RiicSlvRcvCnt ← 0
RiicSlvNum ← in_num
RiicCbSlvTrm ← cbTrm
RiicCbSlvRcv ← cbRcv

Enable the slave address

RIIC0.ICSER register
SAR0E bit ← 1: Slave address in SARL0 and SARU0 is enabled

return
```
5.8.8 User Interface Function (Start a Master Operation)

Figure 5.10 shows the User Interface Function (Start a Master Operation).

Arguments:
- `uint8_t in_addr`: Slave address
- `uint8_t * in_buff`: Pointer to a master transmission/reception data storage
- `uint32_t in_num`: Byte count of master transmissions/receptions
- `CallBackFunc cb`: Callback function (completion of a master transmission/reception)

Flowchart:
1. **Arguments check**: Check if the number of transmit/receive bytes is ≥ 2.
   - Yes → **RIIC_OK**
   - No → **Argument error**

2. **Bus free state check**: Check if the bus is free.
   - Yes → Read the RIIC0.ICCR2 register
     - TRS bit: 0: Receive mode
       → 1: Transmit mode
     - MST bit: 0: Slave mode
       → 1: Master mode
     - BBSY flag: 0: The \( ^{2}\text{C} \) bus is released (bus free state).
       → 1: \( ^{2}\text{C} \) bus is occupied (bus busy state or during bus free time).

3. **Master operation flag set**: Set the master operation flag `RiicMstFlg` to `RIIC_SET`.
4. **Master state set**: Set the master state to busy `RiicMstState` to `RIIC_ST_MST_BUSY`.
5. **Arguments set**: Set arguments in the RAM
   - `RiicTrmAddr ← in_addr`
   - `RiicMstBuff ← in_buff`
   - `RiicMstCnt ← 0`
   - `RiicMstNum ← in_num`
   - `RiicCbMst ← cb`

6. **Number of transmit/receive bytes check**: Check if the number of transmit/receive bytes is equal to 2.
   - Yes → **RIIC_OK**
   - No → Check if master reception

7. **Master reception check**: Check if the master reception is active.
   - Yes → Set the WAIT bit
     - **RIIC0.ICMR3 register**: WAIT bit ← 1
   - No → Issue a start condition
     - **RIIC0.ICCR2 register**: ST bit ← 1

The operation is successfully completed.

Figure 5.10 User Interface Function (Start a Master Operation)
5.8.9 User Interface Function (Obtain the Master State)
Figure 5.11 shows the User Interface Function (Obtain the Master State).

Figure 5.11 User Interface Function (Obtain the Master State)
### 5.8.10 Enabling the RIIC

Figure 5.12 shows the Enabling the RIIC.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RiicEnable</td>
<td>Argument: uint8_t addr: Self-address</td>
</tr>
<tr>
<td>Internal reset</td>
<td>RIIC0.ICCR1 register ICE bit ← 1: Enable (SCL and SDA pins in active state)</td>
</tr>
<tr>
<td>Set the slave address format and slave address</td>
<td>RIIC0.ICSER register SAR0E bit ← 0: Slave address in SARL0 and SARU0 is disabled. RIIC0.SARU0 register FS bit ← 0: The 7-bit address format is selected. RIIC0.SARL0 register ← addr SVA[6:0] bits = addr: Set the self-address.</td>
</tr>
<tr>
<td>Set 100 kbps as the transfer rate</td>
<td>RIIC0.ICMR1 register CKS[2:0] bits ← 2: PCLK/4 clock is selected as the internal reference clock. RIIC0.ICBRH register BRH[4:0] bits ← 24: Set the high-level period of the SCL clock. RIIC0.ICBRL register BRL[4:0] bits ← 29: Set the low-level period of the SCL clock.</td>
</tr>
<tr>
<td>Set the i2C bus mode</td>
<td>RIIC0.ICMR3 register ACKWP bit ← 1: Modification of the ACKBT bit is enabled.</td>
</tr>
<tr>
<td>Set the i2C bus function</td>
<td>RIIC0.ICFER register MALE bit ← 1: Master arbitration-lost detection is enabled.</td>
</tr>
<tr>
<td>Enable or disable interrupts</td>
<td>RIIC0.ICIER register TIE bit ← 1: Transmit data empty interrupt request (ICTXI) is enabled. RIE bit ← 1: Receive data full interrupt request (ICRXI) is enabled. NAKIE bit ← 1: NACK reception interrupt request (NAKI) is enabled. SPIE bit ← 1: Stop condition detection interrupt request (SPI) is enabled. ALIE bit ← 1: Arbitration-lost interrupt request (ALI) is enabled.</td>
</tr>
<tr>
<td>Clear the internal reset</td>
<td>RIIC0.ICCR1 register IICRST bit ← 0: Clear the RIIC reset or internal reset.</td>
</tr>
<tr>
<td>Set port modes</td>
<td>PORT1.PMR register B2 bit ← 1: The P12/SCL pin is used as an I/O port for peripheral functions. B3 bit ← 1: The P13/SDA pin is used as an I/O port for peripheral functions.</td>
</tr>
<tr>
<td>Set the RIIC0 interrupt priority level</td>
<td>IPR246 register IPR[3:0] bits ← 0100b: The RIIC0.EEI0 interrupt priority level is set to level 4. IPR247 register IPR[3:0] bits ← 0100b: The RIIC0.RXI0 interrupt priority level is set to level 4. IPR248 register IPR[3:0] bits ← 0100b: The RIIC0.TXI0 interrupt priority level is set to level 4. IPR249 register IPR[3:0] bits ← 0100b: The RIIC0.TEI0 interrupt priority level is set to level 4.</td>
</tr>
<tr>
<td>Enable the RIIC0 interrupt request</td>
<td>IER1E register IEN6 bit ← 1: The RIIC0.EEI0 interrupt request is enabled. IEN7 bit ← 1: The RIIC0.RXI0 interrupt request is enabled. IER1F register IEN0 bit ← 1: The RIIC0.TXI0 interrupt request is enabled.</td>
</tr>
</tbody>
</table>

**Figure 5.12 Enabling the RIIC**
5.8.11 Disabling the RIIC

Figure 5.13 shows the Disabling the RIIC.

```
RiicDisable

Initialize port modes
PORT1.PMR register
B2 bit ← 0: The P12/SCL is used as a general I/O pin.
B3 bit ← 0: The P13/SDA is used as a general I/O pin.

Clear the internal reset
RIIC0.ICCR1 register
IICRST bit ← 0: Clear the RIIC reset or internal reset.

Transition to the module stop state
PRCR register ← A502h
PRC1 bit = 1: Enable writing to the related registers.
MSTPCRB register
MSTPB21 bit ← 1: The RIIC0 module is in the module stop state.
PRCR register ← A500h
PRC1 bit = 0: Disable writing to the related registers.

return
```

Figure 5.13 Disabling the RIIC
5.8.12 Enabling the RIIC Interrupts

Figure 5.14 shows the Enabling the RIIC Interrupts.

**Figure 5.14 Enabling the RIIC Interrupts**

- **Argument:** uint8_t req: Request enabled
- **RIIC0.ICIER register**
  - RIE bit ← 1: Receive data full interrupt request (ICRXI) is enabled.
  - IER1E register
  - IEN7 bit ← 1
- **RIIC0.ICIER register**
  - TIE bit ← 1: Transmit data empty interrupt request (ICTXI) is enabled.
  - IER1F register
  - IEN0 bit ← 1
- **RIIC0.ICIER register**
  - TEIE bit ← 1: Transmit end interrupt request (ICTEI) is enabled.
  - IER1F register
  - IEN1 bit ← 1
5.8.13 Disabling the RIIC Interrupts

Figure 5.15 shows the Disabling the RIIC Interrupts.

```
RcccIntDis
  | No
  |  | Disabling RXI requested?
  |  |  | Yes
  |  |  | Disable the RIIC0.RXI0 interrupt request
  |  |  | Disable the RXI interrupt and confirm the setting
  |  |  | Clear the RIIC0.RXI0 interrupt request
  |  |  | IER1E register
  |  |  | IEN7 bit ← 0
  |  |  | RIIC0.ICIER register
  |  |  | RIE bit ← 0: Receive data full interrupt request (ICRXI) is disabled.
  |  |  | IR247 register
  |  |  | IR flag ← 0
  |  |  | No
  |  |  | Disabling TXI requested?
  |  |  | Yes
  |  |  | Disable the RIIC0.TXI0 interrupt request
  |  |  | Disable the TXI interrupt and confirm the setting
  |  |  | Clear the RIIC0.TXI0 interrupt request
  |  |  | IER1F register
  |  |  | IEN0 bit ← 0
  |  |  | RIIC0.ICIER register
  |  |  | TIE bit ← 0: Transmit data empty interrupt request (ICTXI) is disabled.
  |  |  | IR248 register
  |  |  | IR flag ← 0
  |  |  | No
  |  |  | Disabling TEI requested?
  |  |  | Yes
  |  |  | Disable the RIIC0.TEI0 interrupt request
  |  |  | Disable the TEI interrupt and confirm the setting
  |  |  | Clear the RIIC0.TEI0 interrupt request
  |  |  | IER1F register
  |  |  | IEN1 bit ← 0
  |  |  | RIIC0.ICIER register
  |  |  | TEIE bit ← 0: Transmit end interrupt request (ICTEI) is disabled.
  |  |  | IR249 register
  |  |  | IR flag ← 0
  |  |  | No
  |  |  | Clear the RIIC0.TXI0 interrupt request
  |  |  | IR248 register
  |  |  | IR flag ← 0
  |  |  | Clear the RIIC0.RXI0 interrupt request
  |  |  | IR247 register
  |  |  | IR flag ← 0
  |  |  | Clear the RIIC0.TEI0 interrupt request
  |  |  | IR249 register
  |  |  | IR flag ← 0
  |  |  | return
```

Figure 5.15 Disabling the RIIC Interrupts
5.8.14 Receive Data Full Interrupt

Figure 5.16 and Figure 5.17 show Receive Data Full Interrupt.

![Flowchart Diagram]

Figure 5.16 Receive Data Full Interrupt (1/2)
return

Yes
No

(AAS0 == 1)

or (STOP == 1)?

Yes

Read the RIIC0.ICSR1 register
AAS0 flag: 0: Slave address 0 is not detected.
1: Slave address 0 is detected.
Read the RIIC0.ICSR2 register
STOP flag: 0: Stop condition is not detected.
1: Stop condition is detected

No

Slave address reception?

Yes

Set the communication flag
RiicStartFlg ← RIIC_SET

No

Dummy read

Read the RIIC0.ICDRR register

Slave reception counter + 1

RiicSlvRcvCnt ← RiicSlvRcvCnt + 1

return

return

return

Figure 5.17 Receive Data Full Interrupt (2/2)
5.8.15  Transmit Data Empty Interrupt

Figure 5.18 shows the Transmit Data Empty Interrupt.

![Diagram of Transmit Data Empty Interrupt]

**Figure 5.18 Transmit Data Empty Interrupt**
5.8.16 Transmit End Interrupt

Figure 5.19 shows the Transmit End Interrupt.

```
Figure 5.19 Transmit End Interrupt

Flowchart:
- **RiicTEND**
  - Clear the TEND flag
  - **RIIC0.ICSR2 register**
    - TEND flag ← 0
  - ** Slave operation**
    - Master operation?
      - Master operation
        - The last data transmission completed?
          - Yes
            - Disabling the RIIC interrupt
              - RiicIntDis()
            - Clear the STOP flag
              - **RIIC0.ICSR2 register**
                - STOP flag ← 0
            - Issue a stop condition
              - **RIIC0.ICCR2 register**
                - SP bit ← 1
          - No
          - return
```
5.8.17 Stop Condition Detection Interrupt
Figure 5.20 shows the Stop Condition Detection Interrupt.

Figure 5.20 Stop Condition Detection Interrupt
### 5.8.18 NACK Detection Interrupt

Figure 5.21 shows the NACK Detection Interrupt.

**Diagram:**

- **RiicNACK**
- **RIIC0.ICIER register** NAKIE bit ← 0
- **Disable the NACK reception interrupt**
- **Disable the RIIC interrupt** RiicIntDis()
- **Slave operation**
- **Master operation**
  - **Master operation?**
  - **Clear the STOP flag** RIIC0.ICSR2 register STOP flag ← 0
  - **Issue a stop condition** RIIC0.ICCR2 register SP bit ← 1
  - **Set the master state** RiicMstState ← RIIC_ST_MST_NACK
  - **Dummy read (SCL released)** Read the RIIC0.ICDRR register
  - **return**
5.8.19 Arbitration-Lost Detection Interrupt

Figure 5.22 shows the Arbitration-Lost Detection Interrupt.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear the AL flag</td>
<td>RiicAL.ICSR2 register AL flag ← 0</td>
</tr>
<tr>
<td>Disabling the RIIC interrupt</td>
<td>Disable the TEI interrupt request</td>
</tr>
<tr>
<td>RiiCintDis()</td>
<td>Enable the RXI interrupt request</td>
</tr>
<tr>
<td>Enabling the RIIC interrupt</td>
<td>Enable the TXI interrupt request</td>
</tr>
<tr>
<td>RiiCintEna()</td>
<td>Clear the master operation flag RiicMstFlg ← RIIC_CLEAR</td>
</tr>
<tr>
<td>Clear the communication flag</td>
<td>RiicStartFlg ← RIIC_CLEAR</td>
</tr>
<tr>
<td>Set the master state</td>
<td>RiicMstState ← RIIC_ST_MST_AL</td>
</tr>
<tr>
<td>Callback function</td>
<td>(completion of master transmission/reception) RiiCcbMst()</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
</tbody>
</table>

Figure 5.22 Arbitration-Lost Detection Interrupt

5.8.20 Timeout Detection Interrupt

Figure 5.23 shows the Timeout Detection Interrupt.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing when timeout is detected</td>
<td>Note: 1. The processing of this function is not included in the sample code. Add a program as required.</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
</tbody>
</table>

Figure 5.23 Timeout Detection Interrupt

5.8.21 Start Condition Detection Interrupt

Figure 5.24 shows the Start Condition Detection Interrupt.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing when a start condition is detected</td>
<td>Note: 1. The processing of this function is not included in the sample code. Add a program as required.</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
</tbody>
</table>

Figure 5.24 Start Condition Detection Interrupt
5.8.22 RIIC0.EEI0 Interrupt Handling

Figure 5.25 shows the RIIC0.EEI0 Interrupt Handling.

```
Excp_RIIC0_EEI0

Verify the interrupt source for STI

Interrupt requested
Read the RIIC0.ICIER register
STIE bit: 0: STI is disabled.
1: STI is enabled.
Read the RIIC0.ICSR2 register
START flag: 0: Start condition is not detected.
1: Start condition is detected.
Start condition detection interrupt
RiicSTART()

Interrupt not requested
Verify the interrupt source for NAKI

Interrupt requested
Read the RIIC0.ICIER register
NAKIE bit: 0: NAKI is disabled.
1: NAKI is enabled.
Read the RIIC0.ICSR2 register
NACKF flag: 0: NACK is not detected.
1: NACK is detected.
NACK detection interrupt
RiicNACK()

Interrupt not requested
Verify the interrupt source for ALI

Interrupt requested
Read the RIIC0.ICIER register
ALIE bit: 0: ALI is disabled.
1: ALI is enabled.
Read the RIIC0.ICSR2 register
AL flag: 0: Arbitration is not lost.
1: Arbitration is lost.
Arbitration-lost detection interrupt
RiicAL()

Interrupt not requested
Verify the interrupt source for TMOI

Interrupt requested
Read the RIIC0.ICIER register
TMOIE bit: 0: TMOI is disabled.
1: TMOI is enabled.
Read the RIIC0.ICSR2 register
TMOF flag: 0: Timeout is not detected.
1: Timeout is detected.
Timeout detection interrupt
RiicTMO()

Verify the interrupt source for SPI

Interrupt not requested
IR = 1
Verify the IR flag
IR = 0

return

Verify the interrupt source for SPI

Interrupt not requested
IR = 1
Verify the IR flag
IR = 0

Stop condition detection interrupt
RiicSTOP()
```

Figure 5.25 RIIC0.EEI0 Interrupt Handling
5.8.23 RIIC0.RXI0 Interrupt Handling
Figure 5.26 shows the RIIC0.RXI0 Interrupt Handling.

```
Excep_RIIC0_RXI0

Receive data full interrupt
RiicRDRF()

return
```

Figure 5.26 RIIC0.RXI0 Interrupt Handling

5.8.24 RIIC0.TXI0 Interrupt Handling
Figure 5.27 shows the RIIC0.TXI0 Interrupt Handling.

```
Excep_RIIC0_TXI0

Transmit data empty interrupt
RiicTDRE()

return
```

Figure 5.27 RIIC0.TXI0 Interrupt Handling

5.8.25 RIIC0.TEI0 Interrupt Handling
Figure 5.28 shows the RIIC0.TEI0 Interrupt Handling.

```
Excep_RIIC0_TEI0

Verify the interrupt source

Interrupt not requested

IR = 1

Verify the IR flag
IR = 0

return

Interrupt requested

Read the RIIC0.ICIER register
TEIE bit: 0: Transmit end interrupt request (ICTEI) is disabled.
1: Transmit end interrupt request (ICTEI) is enabled.

Read the RIIC0.ICSR2 register
TEND flag: 0: Data is being transmitted.
1: Data has been transmitted

Transmit end interrupt
RiicTEND()

Read the IR249 register
IR flag: 0: No interrupt request is generated.
1: An interrupt request is generated.
```

Figure 5.28 RIIC0.TEI0 Interrupt Handling
6. Applying This Application Note to the RX21A or RX220 Group

The sample code accompanying this application note has been confirmed to operate with the RX210 Group. To make the sample code operate with the RX21A or RX220 Group, use this application note in conjunction with the Initial Setting application note for each group.

For details on using this application note with the RX21A and RX220 Groups, refer to “5. Applying the RX210 Group Application Note to the RX21A Group” in the RX21A Group Initial Setting application note, and “4. Applying the RX210 Group Application Note to the RX220 Group” in the RX220 Group Initial Setting application note.
7. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

8. Reference Documents

User’s Manual: Hardware
- RX210 Group User’s Manual: Hardware Rev.1.50 (R01UH0037EJ)
- RX21A Group User’s Manual: Hardware Rev.1.00 (R01UH0251EJ)
- RX220 Group User’s Manual: Hardware Rev.1.10 (R01UH0292EJ)
  The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
  The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
- RX Family C/C++ Compiler Package V.1.01 User’s Manual Rev.1.00 (R20UT0570EJ)
  The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website
  http://www.renesas.com

Inquiries
  http://www.renesas.com/contact/
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Apr. 1, 2013</td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>July 1, 2014</td>
<td>Products: Added the RX21A and RX220 Groups.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Reference Application Notes: Added the Initial Setting application notes for the RX21A and RX220 Groups.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16, 17 Modified the description of reference application note in the following functions: R_INIT_StopModule, R_INIT_NonExistentPort, and R_INIT_Clock.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42 6. Applying This Application Note to the RX21A or RX220 Group: Added.</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable.
   When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal.
   Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the products quality grade, as indicated below.

   - **“Standard”**: Consumer, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment, and industrial robots.

   - **“High Quality”**: Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, and crime systems, and safety equipment, etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implants, etc.) or may cause serious property damages (nuclear reactor control systems, military equipment, etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, the control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used in or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas products or technology described in this document for any purpose relating to military technology, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Notice 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Notice 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

SALES OFFICES

Renesas Electronics Corporation

Refer to “http://www.renesas.com/” for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Nepeanmark, Ontario L3Y 9C3, Canada
Tel: +1-613-898-5441, Fax: +1-613-898-3220

Renesas Electronics Europe Limited
Dukes Meadow Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadionstrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6555-127, Fax: +49-211-6555-1272

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZHCHUN Lu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Toweis, 555 Lingpo Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0588, Fax: +86-21-2226-0595

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 3, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852-2880-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F., No.30, Sec.2, Shuang North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886-2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit 08-02 Hufly Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn. Bhd.
Unit 902, 2nd Floor, Menara Aman, Amanjaya Business Centre, No. 18, Jln Persiaran Barat, 46090 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-5930, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-3141

© 2014 Renesas Electronics Corporation. All rights reserved.

Colophon 4.0